23 Graph Embeddings

Working with large graphs $G = (V, E)$ can be computational expensive, because since there are many combinatorial options to exam. A more recent approach has been to find a way to *embed* the graph in a (often high-dimensional) Euclidean space $\mathbb{R}^d$. By this we mean that:

- each node/vertex $v_i \in V$ is mapped to a point $x_i \in \mathbb{R}^d$
- the relative position $\mathtt{d}_{\cos}(x_i, x_j)$ for any pair of vertices $v_i, v_j \in V$ is reflective of how similar the nodes are in the graph

The specific meaning of the second point is not yet clear, and could mean several things. Different interpretations lead to different formulations. We will start with some we have seen already, an conclude with the basis for the most common approaches.

We will focus on undirected graphs. And assume there are $|V| = n$ vertices.

23.1 Edges as Similarity in Graphs

The first idea is to draw a graph in $\mathbb{R}^d$. If we treat each edge $e = (v_i, v_j)$ as something with similarity $S_{i,j} = 1$, then we can form a similarity matrix $S \in \mathbb{R}^{n \times n}$. Drawing points along the top d eigen-vectors of S is precisely classic MultiDimensional Scaling (cMDS) into $\mathbb{R}^d$.

This is a very reasonable way to capture the connectivity information of a graph, but only gives a very local picture, and is a very blunt form of similarity. While if two nodes v_i, v_j are not directly connected by an edge, it tries to place them far apart – even if there are many other nodes v_k which are connected to both v_i and v_j.

Alternatively, if we first L_1-normalize each of the columns S (note that S is the adjacency matrix), then we get P, a probability transition matrix, of a Markov Chain. We can then consider the top eigenvectors of this. If the Markov Chain is ergodic, this is less intuitive as an embedding. The top-eigenvector places all of the central connected nodes on one side, and the less connected nodes on the other. And, so it is not commonly used for embeddings.

Planar graphs. Given that cMDS is often used to draw data in $\mathbb{R}^2$, we can consider what properties it has. In particular, a *planar graph drawing* is one where vertices are drawn in $\mathbb{R}^2$ and no edges cross. You cannot always do this, and if you can, the graph is called a *planar graph*. Classical MDS does not guarantee that a planar graph will be shown as a planar graph drawing.

A common method to draw low-dimensional graphs, is to put a Spring force between vertices – more if no edge, and less if an edge. Then run a simple simulation of the physics. If each vertex has degree 3, then Tutte's theorem says this will converge to a planar drawing of the graph.

23.2 Laplacian Matrices and Spectral Clustering

In discussion on Spectral clustering, we considered the adjacency matrix $A \in \{0, 1\}^{n \times n}$, and the degree matrix $D \in \mathsf{diag}(n)$, which has the degree of the ith vertex at location $D_{i,i}$ on the diagonal. Recall that I is the identity matrix. Then the *(unnormalized) Laplacian* is defined

$$L_0 = D - A$$

and the *normalized Laplacian* is defined

$$L = I - D^{-1/2}AD^{-1/2} = D^{-1/2}L_0D^{-1/2}.$$

In that context, we considered the top-eigenvectors of L (and L_0). Recall that the second eigenvector of the normalized Laplacian L spread out the data in a way, that helped us find a good normalized cut. This tried to keep the spread-out-ness of vertices controlled (the eigenvector is a unit vector, so the sum of squared values is 1), while stretching edges as little as possible. If there were good clusters (partitions), we could likely find them by scanning vertex subsets in the sorted order. Then recurse on the subsets.

Another common approach is to use the top d eigenvectors, using each as one coordinate for each vertex. With the goal of clustering, this is often part of a two-step process.

Step 1: project on to top d eigenvectors of the normalized Laplacian L.

Step 2: run an algorithm for Euclidean k-means clustering on this embedding into $\mathbb{R}^d$

However, we do not need to run step 2. We can just use the resulting embedding into $\mathbb{R}^d$ as a nice embedding of the vertices in a way that tries to preserve cluster structure (as may be defined by Normalized Cut).

Another place this arises is when we calculate graphs as k-nearest neighbor graphs. Assume that our data $Z \in \mathbb{R}^k$ lies on a nice manifold (this was a very 2005 assumption!). Then we build a graph $G = (V, E)$ where each $z_i \in Z$ becomes a vertex $v_i \in V$. The edges in E are chosen as the k-nearest neighbors of z_i among Z in $\mathbb{R}^k$. Then we use the top eigenvectors of the Laplacian of G to get an embedding of this graph as $X \subset \mathbb{R}^d$. This is called Laplacian Eigenmaps, and was a popular manifold learning / non-linear dimensionality reduction algorithm.

23.3 Random Walk based Embeddings

These approaches have two draw-backs. First, they are somewhat slow, especially if one wants to map to a high-dimensional d; since then one needs to compute d eigenvectors on large graphs. And, so if there is update to the graph data, it may need to be re-calculated. Second, they are still implicitly only looking at edges, and not well-connected neighbors, or what you can get to in a few hops (when a few hops is small, in a very large graph).

Inspired directly by word embeddings (methods of word2vec and GloVe) spawned the idea of what if we measured "neighborhood co-occurrence among random walks" on graphs. Lets unpack this – as explored in papers named *DeepWalk* and *node2vec*.

Consider we took an infinitely long random walk on a connected graph G. Then consider where the path goes through a node v_i. Then we say a nodes v_j are in the L-neighborhood of v_i if that random walk included v_j within L steps before or after v_i on its path. Then we can describe the similarity of nodes v_j and v_i as the probability that a random walk through v_i has v_j within a L-sized neighborhood on the the walk. To be precise, its not quite a probability, and in fact, we can tabulate the number of times it would be in the neighbor on a walk on average (which might be more than 1, especially if L is very large). Think of L as small like $L = 3$ or $= 5$.

Let D_A^{-1} be the operator which L_1-normalizes be degree, so the probability transition matrix P is transformed from the adjacency matrix A as $P = D_A^{-1}A$. The we can define the *co-occurrence matrix* for a random walk of length T (starting from the limiting distribution q_*) as

$$C = 2\sum_{j \in L} \frac{T - j}{|E|} D_A P^j.$$

Note that like in pageRank where we teleport after roughly $1/\beta$ steps, here we perform random walks of limited length.

This captures a more comprehensive notion of similarity between nodes of the graph, for neighbors up to size L, instead of just size 1.

While one could compute C exactly, and use eigenvectors of X. However, this would be slow, since P^L would get quite dense – that is before the eigen-decomposition.

So instead, it is common to use stochastic gradient decent, where a random walk of length T is the batch size. Then we can learn embeddings $x_i \in \mathbb{R}^d$ of nodes v_i that try to respect the similarity encoded in C.

23.4 Warnings about Embeddings

Embeddings, like the graph embeddings, but also word embeddings are now ubiquitous in AI, ML, and Data Mining. They give a nice compact Euclidean representation of objects, which allows for meaningful, easy and fast comparison, classification, and clustering. It is what we kept striving for in this course!

However, it can encode properties that may be unwanted in and unintentional. These are *learned embeddings* but on what data were they trained – is that the truth? The concerns can be seen most clearly with word embeddings, where each word is a single vector in $\mathbb{R}^d$. This is convenient and powerful, but then relations, and correlations can be encoded in ways that might not be desired. Some linear structures can make apparent such correlations. Famously, one can identify an axis between definitionally male and definitionally female words; and this axis was correlated with leadership jobs (engineer, lawyer), versus stereotypically female jobs (secretary, homemaker). While this may reflect historical statistics, it may not be desired in a classifier build on this to predict performance of an applicant (from the resume) on an occupation.